

REMARKS

This paper is submitted in reply to the Office Action dated August 27, 2004, within the three-month period for response (as November 27, 2004 falls on a Saturday). Reconsideration and allowance of all pending claims are respectfully requested.

In the subject Office Action, claims 1, 3-4, 6-10, 13, 15-18, 20, 22-24 and 26-30 were rejected under 35 U.S.C. § 102(b) as being anticipated by U.S. Patent No. 5,845,125 to Nishimura et al. Moreover, claims 2, 5, 14, 19 and 21 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Nishimura et al. in view of U.S. Patent No. 5,093,914 to Coplien et al.; claim 11 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Nishimura et al. in further view of U.S. Patent No. 5,740,440 to West; and claims 12 and 25 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Nishimura et al. in view of Applicant's Admission of Prior Art.

Applicants respectfully traverse the Examiner's rejections to the extent that they are maintained. Applicants have canceled claims 6 and 16, and amended claims 1, 7, 15, 18, 26-28 and 30. Applicants respectfully submit that no new matter is being added by the above amendments, as the amendments are fully supported in the specification, drawings and claims as originally filed.

Now turning to the subject Office Action, and more specifically to the rejection of independent claim 1, this claim generally recites a computer-implemented method of debugging an object-oriented computer program. The method includes setting an inheritance breakpoint that is associated with a first program entity in the object-oriented computer program in which is identified a method in response to user input, and halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a second program entity in the object-oriented computer program that is different from the first program entity.

Claim 1 has also been amended to clarify that the inheritance breakpoint is also associated with the method identified in the first program entity, and that the second program entity "depends from" the first program entity. Claim 6 has been canceled

without prejudice, and claim 7 has been amended, for consistency with the amendments to claim 1.

By reciting that the second program entity “depends from” the first program entity, Applicants have clarified that the second program entity has a dependent relationship relative to the first program entity, as opposed to a base or superclass relationship. As such, where the first program entity is a class, and the method in the first program entity is an implementation of the method, the second program entity may be a subclass of the first program entity, and the implementation of the method in the second program entity may be an overriding implementation of the method. Likewise, where the first program entity is an interface or an abstract class, the second program entity may be a class wherein a method identified in the interface or abstract class is ultimately implemented. By doing so, a user is able to specify a given method to associate with a breakpoint, and then have a debugger halt execution whenever any implementation of that method is hit.

Also, by clarifying that an inheritance breakpoint is associated with a method, Applicants have clarified the aspect of claim 1 whereby execution may be halted on one or more implementations of that method in a dependent class or other program entity.

Claim 1 has been rejected as being anticipated by Nishimura, which discloses a debugger for an object-oriented environment. However, Nishimura is specifically directed to setting breakpoints on individual objects, or instances, of a class, rather than upon all instances of a class. Furthermore, Nishimura is directed to addressing the problem that arises due to inheritance, whereby a given instance of a class that inherits from a parent or base class may have certain methods and data that are defined in the base class, rather than the specific class for the instance, which for prior designs required a user to set individual breakpoints in a class and any of its base classes to ensure that a debugger will break on a desired object. (col. 4, lines 9-33).

Nishimura addresses this problem through the use of “object-type” breakpoints, which results in the automated setting of breakpoints in all of the public functions of a class and of any base (parent) classes therefor (col. 15, lines 11-13, and Fig. 9). Of note,

Page 11 of 16
Serial No. 09/998,511
Reply to Office Action of August 27, 2004
IBM Docket ROC920010096US1
WH&E IBM/194

this mechanism in Nishimura for automatically setting breakpoints in the base or parent classes for a given object so that a program will be halted when a breakpoint in a base class is encountered is precisely the opposite scenario to that recited in claim 1.

Specifically, claim 1 recites halting execution at an implementation of a method that is defined in a second program entity that depends from a first program entity with which an inheritance breakpoint is associated, while Nishimura sets breakpoints in parent or base classes of a given object.

By way of example, consider a program including three classes, A, B and C, where class B is a subclass of class A, and class C is a subclass of class B. Consider also that class B has a method X, and class C overrides this method with a method X'. Using the invention recited in claim 1, a user may be permitted to set an inheritance breakpoint on method X of class B, and have the program halted upon encountering method X' in class C. Conversely, in the Nishimura system, a user may be permitted to set a breakpoint on an instance of class B, and have a breakpoint automatically set in class A. Given that class C depends from class B, however, Nishimura would not automatically set any breakpoint on an instance of class C as a result of setting the breakpoint on the instance of class B.

Given, therefore, that Nishimura discloses the opposite configuration to claim 1, Nishimura does not disclose halting execution in response to reaching an implementation of a method defined in a second program entity that depends from a first program entity with which an inheritance breakpoint is associated. Claim 1 is therefore novel over Nishimura.

Claim 1 is also non-obvious over Nishimura, as the reference is directed to solving a completely different problem, and as the solution proposed by Nishimura results in precisely the opposite configuration to that recited in claim 1. Reconsideration and allowance of claim 1, as well as of claims 2-5 and 7-12 which depend therefrom, are respectfully requested.

Next with respect to independent claim 13, this claim generally recites a computer-implemented method of debugging an object-oriented computer program. The

method includes setting an inheritance breakpoint that is associated with a first class in the object-oriented computer program in which is identified a method in response to user input, and halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a second class in the object-oriented computer program that inherits from the first class.

Claim 13 is rejected as being anticipated by Nishimura. However, as discussed above in connection with claim 1, Nishimura discloses automatically setting breakpoints in parent or base classes of a given class instance. Claim 13, however, discloses an opposite scenario where execution is halted in response to reaching an implementation of a method in a class that inherits from another class with which an inheritance breakpoint is associated. Applicants therefore respectfully submit that claim 13 is novel over Nishimura. Furthermore, claim 13 is non-obvious over the reference as there is no suggestion in Nishimura of the desirability of halting execution in a subclass of a particular class with which a breakpoint is associated. Reconsideration and allowance of claim 13 are therefore respectfully requested.

Next with respect to independent claim 14, this claim generally recites a computer-implemented method of debugging an object-oriented computer program. The method includes setting an inheritance breakpoint that is associated with an interface in the object-oriented computer program in which is identified a method in response to user input, and halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a class in the object-oriented computer program that implements the interface.

Claim 14 is rejected as being obvious in view of Nishimura and Coplien. As noted above in connection with claim 1, however, Nishimura discloses setting breakpoints in base or parent classes of a given class instance, but does not disclose halting execution upon reaching a program entity that is dependent upon another program entity. Likewise, Nishimura does not disclose the concept of setting a breakpoint that is associated with an interface and halting execution upon reaching an implementation of a

method defined in a class that implements the interface, and indeed, the Examiner admits as such in paragraph 4 of the subject Office Action.

The Examiner relies on Coplien for allegedly disclosing an interface/implementation relationship in object-oriented programming. However, Coplien merely discloses this general concept in object-oriented programming, but falls short of disclosing halting execution of a program upon reaching an implementation of a method identified in an interface with which a breakpoint is associated. What Coplien does disclose is a technique similar to Nishimura, whereby a breakpoint is set on a specific instance of an object, when the address of a function defined in the instance is not known until runtime.

Given also that Nishimura discloses setting breakpoints on base or parent classes, rather than in dependent entities such as implementations of interfaces, Applicants also submit that one of ordinary skill in the art would not be motivated to modify Nishimura to set breakpoints on dependent entities such as interface implementations. Reconsideration and allowance of claim 14 are therefore respectfully requested.

Next with respect to independent claim 15, this claim generally recites a computer-implemented method of debugging an object-oriented computer program. The method includes receiving user input to halt program execution during debugging in response to reaching any of a plurality of implementations of a method in an object-oriented computer program, and thereafter setting a breakpoint for at least a subset of the plurality of implementations such that execution of the object-oriented computer program will be halted in response to reaching any of the implementations on which a breakpoint has been set.

Claim 15 has also been amended to clarify that the user input to halt program execution includes user input to set an inheritance breakpoint on the method, that the inheritance breakpoint is associated with a first program entity, and that at least one of the plurality of implementations of the method is defined in a second program entity that depends from the first program entity. The claim has also been amended to clarify that the subset of implementations upon which a breakpoint is set includes the implementation

defined in the second program entity. Claim 16 has also been canceled for consistency with the amendments to claim 15.

As discussed above in connection with claim 1, Nishimura discloses only the setting of breakpoints in base or parent classes of a given instance of a class. As claim 15 recites automatically setting a breakpoint in a program entity that depends from another program entity (e.g., a subclass or an implementation of an interface), claim 15 is an opposite scenario to that disclosed in Nishimura. Claim 15 is therefore novel and non-obvious over Nishimura, and reconsideration and allowance of claim 15, and of claim 17 which depends therefrom, are respectfully requested..

Next with respect to independent claims 18 and 28, each of these claims has been amended in a similar manner to claim 1, and now recite the concept of halting execution upon reaching an implementation of a method defined in a second program entity that depends from a first program entity with which is associated an inheritance breakpoint. As discussed above in connection with claim 1, this combination of features is not disclosed or suggested by the prior art of record. Reconsideration and allowance of claims 18 and 28, and of claims 19-25 and 29 which depend therefrom, are therefore respectfully requested.

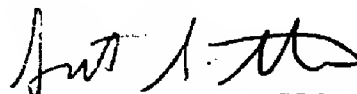
Likewise, with respect to independent claims 26 and 30, each of these claims has been amended in a similar manner to claim 15 (claim 27 has also been amended for consistency therewith), and these claims now recite the concept of setting a breakpoint on an implementation of a method in a second program entity that depends from a first program entity with which an inheritance breakpoint is associated. Claims 26 and 30 are therefore novel and non-obvious over the prior art of record for the same reasons presented above for claim 15. Reconsideration and allowance of claims 26 and 30, and of claim 27 which depends therefrom, are respectfully requested.

In summary, Applicants respectfully submit that all pending claims are novel and non-obvious over the prior art of record. Reconsideration and allowance of all pending claims are therefore respectfully requested. If the Examiner has any questions regarding the foregoing, or which might otherwise further this case onto allowance, the Examiner

may contact the undersigned at (513) 241-2324. Moreover, if any other charges or credits are necessary to complete this communication, please apply them to Deposit Account 23-3000.

29 NOV 2004
Date

Respectfully submitted,



Scott A. Stinebruner
Reg. No. 38,323
WOOD, HERRON & EVANS, L.L.P.
2700 Carew Tower
441 Vine Street
Cincinnati, Ohio 45202
Telephone: (513) 241-2324
Facsimile: (513) 241-6234